A Runtime Evaluation Methodology and Framework for Autonomic Systems

Chonghyun Lee[1], Hyunsang Youn[1], Ingeol Chun[2] and Eunseok Lee[1]
[1] Sungkyunkwan University /Department of Computer Engineering, Suwon, South Korea
Email: carve9142@skku.edu, wizehack@skku.edu, leees@skku.edu
[2]Electronics and Telecommunications Research Institute(ETRI), Daejeon, South Korea
Email: igchun@etri.re.kr

Abstract - **An autonomic system provides self-adaptive ability that enables system to dynamically adjust its behavior on environmental changes or system failure. Fundamental process of adaptive behavior in an autonomic system is consist of monitoring system or/and environment information, analyzing monitored information, planning adaptation policy and executing selected policy. Evaluating system utility is one of a significant part among them. We propose a novel approach on evaluating autonomic system at runtime. Our proposed method takes advantage of a goal model that has been widely used at requirement elicitation phase to capture system requirements. We suggest the state-based goal model that is dynamically activated as the system state changes. In addition, we defined type of constraints that can be used to evaluate goal satisfaction level. We implemented a prototype of autonomic computing software engine to verity our proposed method. We simulated the behavior of the autonomic computing engine with the home surveillance robot scenario and observed the validity of our proposed method**

Index Terms—**autonomic computing, adaptive systems, goal model, embedded system**

I. Introduction

As computing systems runtime environment is becoming extremely complex, unpredictable situations those were unforeseen at system design phase frequently occurs at runtime. To cope with such situations after system deployment, enormous cost might be required, especially for the large scale embedded systems such as a cyber physical system(CPS). Autonomic computing has been proposed to deal with such issue in pursuit of developing software that adapt to its own situation and minimize the occurrence of the system failure without human intervention. Such autonomic systems usually consist of system properties that deliver a monitoring, analysis, planning and execution feedback loop. Evaluation of the autonomic system is one of a crucial part among various research area related to autonomic computing. If an autonomic system provides a wrong evaluation of the adaptation result, such system might continuously make a wrong decision. Despite such importance, there are not enough leading research and actual practice. In this paper, we propose a novel approach on evaluating autonomic system at runtime. Our proposed method exploits a goal model that has been widely used at requirement elicitation phase to capture system requirements. To utilize a goal model on runtime evaluation of the autonomic system, we defined constraints that can be regarded as a runtime attribute of the goal model. Furthermore, to reduce the message transmission overhead, our goal model is dynamically activated depending on the system state.

This paper is organized as follows: the next section provides a brief introduction of autonomic computing and other related researches. Section III presents our self-adaptive autonomic system framework and autonomic systems runtime evaluation methodology. Section c! presents the implementation details of our prototype autonomic system and evaluation results. In section d!, we conclude our work with future research directives.

II. Backgrounds

A. Autonomic Computing

Autonomic computing (AC) is a computing paradigm proposed by IBM[5] to solve the problems associated with the complex computing systems and unpredictable operational environment. In conventional computing paradigm, software engineers design, implement and test the software assuming that the system will be operated under predictable runtime environment predefined at requirement elicitation process. As system execution environment have become extremely complex, it is impossible to predict every situation a system might undergo. Autonomic computing introduced the concept of the human autonomic nervous system, which controls vital bodily function without human intervention. Autonomic computing allows such functionalities to the system by embedding additional infrastructure in the system. The core aspect of autonomic computing systems is as below:

Self-configuration – Automatically re-configuring system components or integrating new components as system policy.

Self-optimization – Continuously attempt to improve system performance and efficiency seamlessly.

Self-healing – Automatically detects, localize system failure and recovers software and hardware problems

Self-protection – System defends itself from malicious attacks or failures.

B. Utility-based Self-adaptive System Evaluation

Self-adaptive systems change system behavior or structures without human operator involvement. Therefore, deciding when to execute adaptive policies and evaluating how effectively applied policies actually improved system utilities are crucial. Kephart[9] suggested a system utility evaluation method for self-adaptive system. The author separated self-adaptive system utility as service level utility and resource level utility. The method calculates every possible resource level utility to find the best amount of resource to be allocated. D. Garlan[10][11] also proposed similar approach on calculating system utility in Rainbow self-adaptive software architecture. Such methods were applied to a large-scale data center and web server system respectively to show validity. Both of aforementioned methods adopt limited number of metric to calculate overall system utility such as response time. However, such method is limited to a system with multiple objectives such as a robotics system, since a robotics system generally offers various functions.

C. Goal Oriented Requirement Engineering

A goal model has been used to capture system requirements for traditional systems on requirement elicitation phase. Goals tend to reflect what the system is supposed to do, and it has been regarded as a natural way to model system requirements.

The Knowledge Acquisition in automated Specification(KAOS) framework is a well-known approach for modeling system requirements using a goal model[12]. The KAOS model is used at requirement elicitation phase to capture system goals. System goals are specified at high-level goals, and each of them is then decomposed into sub-goals that elaborate how the goal is achieved. The i* framework is another goal-oriented requirement engineering approach that focus on the interaction between actors. There are several recent researches that extend existing approaches to deal with uncertainty or inconsistency. However, as far as we know there is no approach that takes advantage of a goal model to evaluate autonomic systems behavior and the result of adaptation at runtime.

III. Proposed approach

In this section, we first give a brief overview of our autonomic computing software framework. Then the detailed explanation about how the autonomic computing engine's analysis and evaluation are performed will be followed.

A. Overall Self-adaptive System Framework

Fig. 1 shows an overall structure of our autonomic computing software engine. It is an improved version of our previous work[3]. We originally exploited rule-based tables as knowledge models, but we replaced these with tree structured goal models, and a fault tree.

The autonomic computing engine in Fig. 1 interacts with the managed element, namely the actual running system, through the sensor and the effecter. The autonomic computing engine continuously checks the system status at runtime, and decides whether adaptation policies are required. A detailed functionality of each module in the autonomic computing engine is as follows:

Monitoring module – The monitoring module requests required data to the managed element and receives the data through the sensor. Monitored data are then reported to the analysis module to check goal violation.

Analysis module – The analysis module examines received data based on the analysis model(goal model) in the knowledge base. Analysis model actually computes goal satisfaction level at runtime. The analysis module sends goal violation signal to the diagnosis module when goal satisfaction level is below the predefined threshold. A detailed process of calculating goal satisfaction level is described in section III - B.

Diagnosis module – The diagnosis module receives goal violation information from the analysis module. Failure information from the analysis module is then mapped into the top event of the diagnosis model, which is represented as a fault tree in our framework. Fault tree is a graph representing the combination of the fault events that can derive specific failure of a system. Conventional fault tree is used at system design phase to analyze system reliability, but we utilize a fault tree at runtime to localize the root cause of the failure. Since the detailed mechanism of the runtime fault tree is out of this papers scope, we do not describe it in this paper.

Planning Module – The planning module receives root cause information from the diagnosis module. The planning module choose corresponding plan among several alternative plans and pass the chosen policy id to the execution module. The execution module then sends policy id to the managed elements through the effecter. After policy execution, the evaluation module receives constraint information again from the managed elements and sends calculated goal satisfaction level to the planning module for feedback.

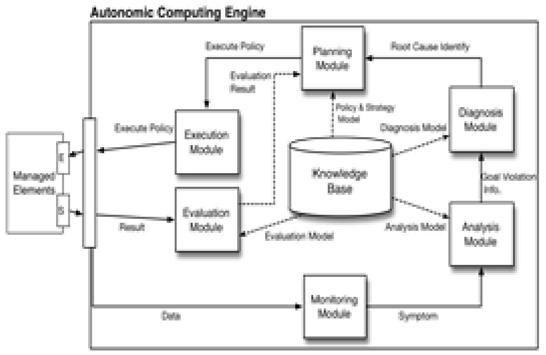


Figure 1 Overall autonomic computing engine architecture

B. Runtime evaluation

To evaluate system status at runtime, we exploit a goal model. By using a goal model, a system can trace current system status so that deciding which data to monitor becomes easier. In case of a simple system such as a web server, only limited metrics such as response time and server cost are enough to monitor the system performance. On the other hand, a system that supports complex services such as robotic systems, an evaluation metric should be changed considering current system status or objectives that the system is supposed to achieve. The basic structure of our goal model is similar with the conventional goal model such as the one used in KAOS. Fig. 2(a) shows an example goal model of our approach. The model represents the basic requirements of a home surveillance robot that is capable of move to the destination, avoiding obstacles, taking a picture with embedded camera, reporting environment information such as humidity and temperature, and constant battery checking. Upper level goals(parent node) are decomposed into lower level goals(child node) and each child has a weight depending on the contribution level to a parent goal. Each weight can be gained by using weight elicitation methods in the value tree analysis [14].

At runtime, an entire goal model shown in fig. 2(a) is not fully maintained. Instead, the goals those are related to current system state are activated at runtime. For example, when a robot's current goal is to move to some location (*Move* state), activated goals at this time will be G0, G1, G4, G5, G6 and G12, as shown in fig. 2(b). The robot is supposed to avoid obstacles in the moving path, so when the robot finds an obstacle, the system state will be changed from *Move* state to *Avoid Obstacles* state. Therefore, the goals corresponding to *Avoid Obstacles* state will be automatically activated as the system state changes (fig. 2(c)). Each leaf goal has one or more constraints that can be calculated at runtime. There are two kinds of constraints in our model; maintenance constraints(MC) and operation constrains(OC).

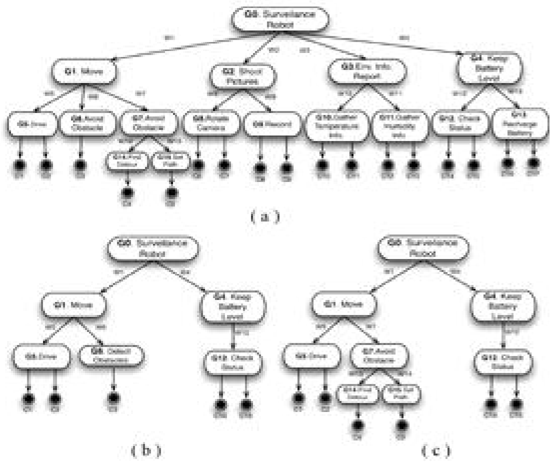


Figure 2 Example goal models

MC represents a desire to keep a condition true over entire operation time. For example, when a robot is not supposed to stop during the *Move* state, a MC could be *keep the speed over 0.0*(or some other threshold value). MCs are periodically monitored during runtime at certain time interval, and the analysis module captures the goal violation as soon as MCs are violated. OCs represents the constraints those can be evaluated after execution of the goal. For example, when a requirement for a robot is to *reach to a certain destination*, OC becomes *elapsed time t* to reach to the destination. On evaluating OCs, we adopted fuzzy concept to deal with the ambiguity around the constraint boundary. For instance, if aforementioned OC is set to return 0.0 of satisfaction level after time t, the result will be 0.0 no matter how close the robot accomplished the goal to the requirement, and unnecessary adaptation process will continuously take place. Adopting fuzzy concept to a goal model is initially proposed by Baresi[8], but they only applied it to non-functional requirement in the goal model. In our perspective, providing flexibility on evaluating functional requirements is also necessary in many cases, so we extended fuzzy concept to every leaf goals. To measure the satisfaction level of operation constraints, a membership function should be set. Fiq. 3 show examples of membership functions.

The evaluation module in fig. 1 calculates the total goal satisfaction level. Each activated leaf goal's achievement level(*LGA*) is measured with the equation below:

$$\mathrm{LGA}(G_k) = \mathrm{MC}_k \cdot (\sum_{i=1}^{n} \mathrm{w}_i \cdot \mathrm{f}_{ki}(\mathrm{OC}_{ki}) \qquad (1)$$

where G_k denotes k th goal in the goal model and MC_k, OC_{ki} is the satisfaction level of the maintenance constraint and the operation constrain respectively. f_{ki} is i th operation constraint's membership function, where n is the number of the operation constrains, so the operation constraints are weighted sum of each constraint. After calculating LGAs, overall achievement level of the current goal model is also calculated with weighted sum method. If a goal node has one or more leaf goals, equation(2) is applied, otherwise equation (3) is recursively calculated until it reach to the root goal.

$$\mathrm{GA}(G_k) = \sum_{i=1}^{n} w_i \cdot LGA(G_i) \qquad (2)$$

$$\mathrm{GA}(G_k) = \sum_{i=1}^{n} w_i \cdot GA(G_i) \qquad (3)$$

IV. Evaluation

A. Scenario

Due to the lack of freely available open source autonomic systems, we implemented a prototype of an autonomic computing engine to validate the feasibility of our proposed method(fig. 4). Our prototype is based on the home surveillance robot, which has six available states; *start, idle, move, avoid obstacle, recharge, environment information report, shoot pictures* and *end*. The autonomic computing engine was developed in Java using Eclipse SDK, along with SQLiteJDBC a Java driver for SQLite. In this prototype system, the autonomic computing engine is not embedded in the managed element, so we took advantage of TCP/IP protocol for the communication between the managed element and the autonomic computing engine. This protocol can be replaced with inter-process communication(IPC) when the autonomic engine is embedded into the managed element. To demonstrate the home surveillance robot, we implemented a simulated home environment. The robot consists of eleven sensors that can be monitored by autonomic computing engine. Our first experiment was set to evaluate whether the system accurately evaluates the system utility and detects goal violation. We simulated the robots moving behavior that belongs to the move state. The robot continuously sent constraint information(current velocity in this case) to the autonomic computing engine when moving from the starting point to the destination point. When the robot reached to the destination, it sent operation constraint information to the autonomic computing engine to calculate goal satisfaction level. To validate our state-based goal model in terms of efficiency, we also simulated each available state and counted the number of messages sent between the home surveillance robot and the autonomic computing engine. For the counterpart, we counted the number of messages when the full goal model is kept during operation.

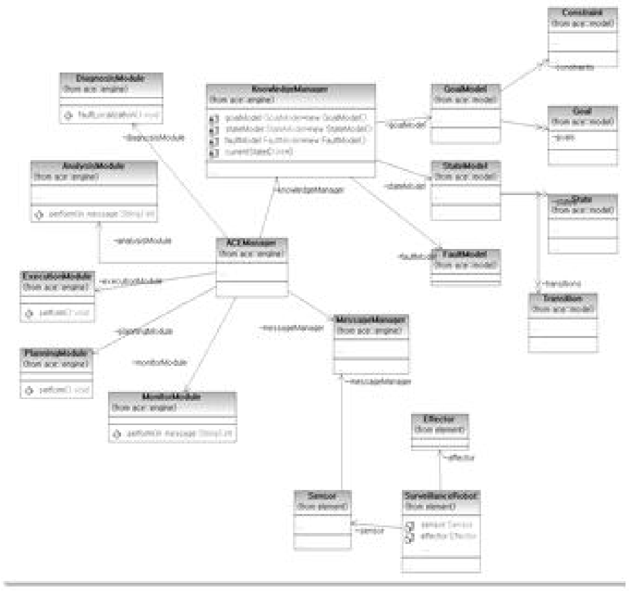
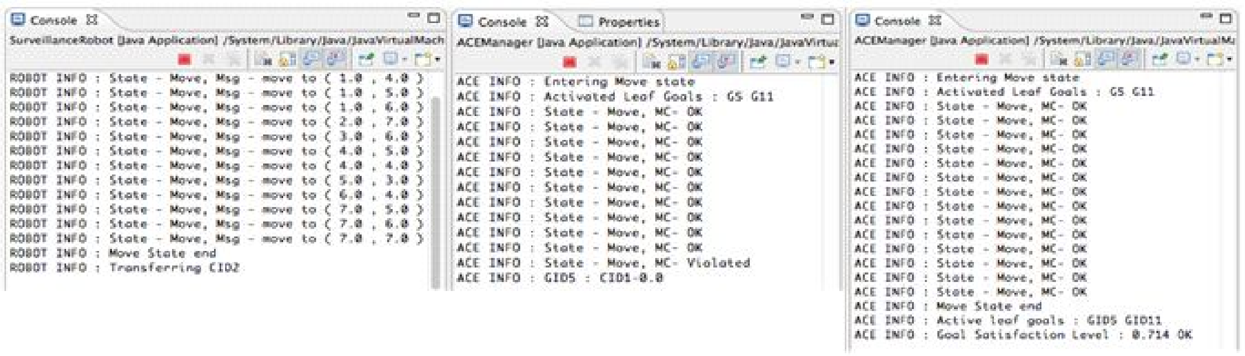


Figure 4 Class diagram of the autonomic computing engine

B. Experimental Result

Fig. 6 shows the operation log message of the robot and the autonomic computing engine. In fig. 6(a), the surveillance robot continuously sent maintenance constraint information to the autonomic computing engine during move operation. As soon as the robot arrived at the destination, operation constraint information(constraint ID(CID2-elapsed time)) was sent to the autonomic computing engine. In fig. 6(b), we simulated that the robot was stalled on the middle of path, and CID1(current velocity) became 0.0, so the autonomic computing engine instantly detected the goal violation. In fig. 6(c), the robots move behavior ended without any goal violation, so the autonomic computing engine received the operation constraint information from the robot and calculated goal satisfaction level based on current active goal tree(fig. 2(b)). We set the elapsed time to 7.0 seconds, where the designated constraint on goal model to finish move operation was 6.0seconds, therefore the membership function shown in fig. 3(a) returned 0.714 of satisfaction level.

In our second set of experiments, we calculated total number of messages sent between the autonomic computing engine and the home surveillance robot to measure the cost of monitoring. We simulated the autonomic computing engine's move, obstacle avoid, environment information report, shoot pictures and charge battery state with state-based goal model that we propose.



```
SurveillanceRobot [Java Application] /System/Library/Java/JavaVirtualMach
ROBOT INFO : State - Move, Msg - move to ( 1.0 , 4.0 )
ROBOT INFO : State - Move, Msg - move to ( 1.0 , 5.0 )
ROBOT INFO : State - Move, Msg - move to ( 1.0 , 6.0 )
ROBOT INFO : State - Move, Msg - move to ( 2.0 , 7.0 )
ROBOT INFO : State - Move, Msg - move to ( 3.0 , 6.0 )
ROBOT INFO : State - Move, Msg - move to ( 4.0 , 5.0 )
ROBOT INFO : State - Move, Msg - move to ( 4.0 , 4.0 )
ROBOT INFO : State - Move, Msg - move to ( 5.0 , 3.0 )
ROBOT INFO : State - Move, Msg - move to ( 6.0 , 4.0 )
ROBOT INFO : State - Move, Msg - move to ( 7.0 , 5.0 )
ROBOT INFO : State - Move, Msg - move to ( 7.0 , 6.0 )
ROBOT INFO : State - Move, Msg - move to ( 7.0 , 7.0 )
ROBOT INFO : Move State end
ROBOT INFO : Transferring CID2
```

```
ACEManager [Java Application] /System/Library/Java/JavaVirtua
ACE INFO : Entering Move state
ACE INFO : Activated Leaf Goals : G5 G11
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- Violated
ACE INFO : GID5 : CID1-0.0
```

```
ACEManager [Java Application] /System/Library/Java/JavaVirtualM
ACE INFO : Entering Move state
ACE INFO : Activated Leaf Goals : G5 G11
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : State - Move, MC- OK
ACE INFO : Move State end
ACE INFO : Active leaf goals : GID5 GID11
ACE INFO : Goal Satisfaction Level : 0.714 OK
```

(a) (b) (c)

Figure 5 Samples of runtime evaluation

TABLE 1 NUMBER OF MESSAGES BETWEEN AUTONOMIC COMPUTING ENGINE AND THE HOME SURVEILLANCE ROBOT

State	Avg. number of messages	Reduced ratio(%)
Move	53.56	9.77
Obstacle avoid	41.27	7.53
Env.info. report	151.23	27.58
Shoot pictures	91.22	16.63
Charge battery	211.12	38.50

At the same time, we simulated the autonomic behavior with a goal model that fully maintains state information. As expected, the result shown in table.2 indicates that for any possible states, our proposed method reduced the number of messages. On average, the number of messages passed was reduced 20% than its counterpart. This is due to our goal model that dynamically activates goals as system status changes. Since unnecessary goals in current state are excluded, the number of messages could be reduced.

CONCLUSIONS

We have proposed a runtime evaluation method for the autonomous system. Our evaluation method takes advantage of stated-based goal model that dynamically activates the goal model depending on the system state. Through a simulated prototype system, we have observed that our proposed method can properly evaluate system utility and detect system failures. In addition, our experiment has shown that the monitoring overhead from message passing was reduced by 20% with our state-based goal model.

For the future work, we are investigating ways to dynamically changing weights in the goal model, depending on the contextual situation. This would provide the autonomic system a more robust way to evaluate system utility in various environments.

Acknowledgment

This work was supported by the IT R&D program of MKE/KEIT [10035708, “The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software”].

References

[1] C. Zhong, S. A. DeLoach, “Runtime models for automatic reorganization of multi-robot systems”, In Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems (SEAMS 2011). ACM, New York, NY, USA, 20-29.
[2] Emmanuel Letier, Axel van Lamsweerde, “Reasoning about partial goal satisfaction for requirements and design engineering”, In Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering (SIGSOFT ’04/FSE-12), pp. 53-62, 2004
[3] GiljongYoo, HyunsangYoun, Eunseok Lee, “Design methodology for runtime self-adaptiveness of CPS”, International Conference on Computers, Communications, Control and Automation, vol.2, pp. 82-85, 2011
[4] Jae Sun Kim, Sooyong Park, VijayanSugumaran, “Contextual problem detection and management during software execution in complex environments”, Industrial Management & Data Systems, vol. 106, issue. 4, pp. 540-561. 2006
[5] J. O. Kephart, “Achieving self-management via utility functions”, IEEE Internet Computing – INTERNET, vol. 11, no.1, pp.40-48, 2007
[6] J. O. Kephart, D. M. Chess, “The vision of autonomic computing”, IEEE Computer – COMPUTER, vol. 36, no. 1, pp.41-50, 2003
[7] Luciano Baresi, Liliana Pasquale, “Adaptive goals for self-adaptive service compositions”, IEEE International Conference on Web Services(ICWS 2010), pp. 353-360, 2010
[8] Luciano Baresi, Liliana Pasquale, Paola Spoletini, “Fuzzy goals for requirements-driven adaptation”, In Proceedings of the 2010 18th IEEE International Requirements Engineering Conference (RE 2010), pp. 125-134, 2010
[9] S. A. DeLoach, M. Miller, “A goal model for adaptive complex systems”, International Journal of Computational Intelligence: Theory and Practice, vol. 5, no. 2, 2010
[10] Shang-Wen Cheng, David Garlan, Bradley R. Schmerl, “Evaluating the effectiveness of the Rainbow self-adaptive system”, International Conference on Software Engineering – ICSE, pp. 132-144, 2009
[11] Shang-Wen, An-Cheng Huang, David Garlan, Bradley R. Schmerl, PerterSteenkiste, “Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure”, IEEEComputer – COMPUTER, vol. 37, no. 10, pp.276-277, 2004
[12] Van Lamsweerde, A. and Letier, “handling obstacles in goal-oriented requirements engineering”, IEEE Trans. On Software Engineering. 26(10): 978-1005
[13] W. E. Walsh, G. Tesauro, J. O. Kephart, R. Das, “Utility functions in autonomic systems”, International Conference on Autonomic Computing – ICAC, pp. 70-77, 2004
[14] www.mcda.hut.fi/value_tree/theory/theory.pdf